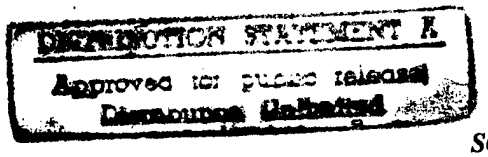
DEPARTMENT OF THE AIR FORCE
HQ Air Force Operational Test and Evaluation Center

AFOTEC PAMPHLET 99-102, Volume 6
1 MARCH 1996



Test and Evaluation

SOFTWARE MATURITY EVALUATION GUIDE

The purpose of this pamphlet is to provide Air Force Operational Test and Evaluation Center (AFOTEC) personnel information needed to evaluate software maturity. This pamphlet describes AFOTEC's software maturity evaluation concept, the data requirements to support this evaluation, planning considerations, evaluation instructions, and guidance on reporting a software maturity evaluation.

This volume is number six in a series of software operational test and evaluation guides prepared by the Software Analysis Team (SAS) at Headquarters (HQ) AFOTEC. Local reproduction of all volumes in this series is authorized. This volume is an evolutionary document that will be updated periodically. Comments should be directed to the office of primary responsibility (OPR).

SUMMARY OF CHANGES

AFOTEC Pamphlet 99-102, volume 6, replaces AFOTEC Pamphlet 800-2, volume 6. This document has been completely rewritten.

Supersedes AFOTECP 800-2, Vol 6, 1 Oct 90
OPR: SAS (Capt Brian Hermann)

Please Recycle Printed on Recycled Paper

Certified by: SA (Col A. W. Groves)
Pages: 33 ~~Distribution: F~~

Chapter 1

INTRODUCTION

1.1. General. This pamphlet describes how to plan, conduct, and report a software maturity evaluation in support of AFOTEC-conducted operational test and evaluation (OT&E). This guide includes sample maturity charts to illustrate evaluation techniques.

1.2. Overview of the Guide. This guide is organized as follows:

Chapter 1 - Provides background information and definitions related to software maturity and outlines planning requirements for a maturity evaluation.

Chapter 2 - Describes software maturity data, sources, and collection time frame.

Chapter 3 - Outlines the evaluation process by presenting sample maturity charts and describing software maturity reporting.

Chapter 4 - Provides lessons learned from past evaluations to answer frequently asked questions.

Attachment 1 - Sample measure of effectiveness (MOE). Use this example as a basis for providing software maturity inputs to test and evaluation master plans (TEMP).

Attachment 2 - Software maturity data definitions. This list of data items helps the evaluator determine what information is required for specific evaluation functions.

Attachment 3 - Description of software severity levels. These definitions should be used to standardize the severity level assignment process in each program.

Attachment 4 - Sample software maturity data request letter. This sample letter outlines information the evaluator should request to perform a maturity evaluation.

Attachment 5 - Sample software maturity evaluation report outline. This outline provides a simple framework for most maturity reports.

Attachment 6 - This attachment presents an explanation of how to evaluate sample and ideal maturity charts and trends.

Attachment 7 - Definition of acronyms and abbreviations.

1.3. Overview of Software Maturity Evaluation.

Software maturity is a measure of the progress the software products are making towards satisfying user requirements.

1.3.1. **What is software maturity and why is it important?** Software maturity is a measure of the progress the software products are making towards satisfying user requirements. AFOTEC uses this metric to support operational test readiness decisions (software's ability to support the rigors of operational testing) and actual OT&E. Our current software maturity evaluation techniques do not project change trends or test readiness, but we expect to add that capability in future versions of this document.

1.3.2. **Why do we evaluate software maturity before OT&E begins?** Software maturity evaluations aid the decision-maker in answering the question **"Is the software ready for test?"** Data collected during system integration and test are used to prepare various demonstrated failure/change trends which the software test manager (STM) and deputy for software evaluation (DSE) use to evaluate software maturity. The underlying philosophy of maturity is: *the RATE and SEVERITY of software changes necessary to support new user requirements or correct errors should decrease over time.* A software system that demonstrates these decreasing trends indicates maturity or progress toward maturity. Refer to figure 1.1 for an example of the "idealized." graphical presentation of a mature/maturing system when change data are displayed over time. These demonstrated changes may occur for any of the following reasons:

- to correct errors (corrective change)
- to enhance system capability (perfective change)
- to make the software compatible with changes in the computing environment (adaptive change)

Software maturity is concerned with all of these changes but the typical problems found and changes requested during OT&E should be predominantly corrective with fewer adaptive and perfective changes.

1.3.3. **If we evaluate software maturity before OT&E, why do we continue during OT&E?** Our methodologies are progressing toward evaluating software reliability during OT&E, but many programs don't have the type or quantity of data required to evaluate software reliability. If we can't evaluate reliability, it is valuable to present a final snapshot of the software's maturity for decision-makers, future users, and future maintainers before AFOTEC's involvement ends. In addition, the software maturity data collected during OT&E are more operationally representative than data collected during developmental testing. Because the data are more realistic, the evaluator may choose to place more emphasis on the software problems found during OT&E.

1.3.4. **Where can I find further software maturity policy guidance?**

- Air Force Instruction (AFI) 10-602, *Determining Logistics Support and Readiness Requirements,* defines software maturity and suggests that major commands include maturity as an operational requirement.
- Secretary of the Air Force for Acquisition (SAF/AQ) Policy Letter 93M-017, *Software Metrics,* provides for the establishment of a data collection system to support software maturity evaluations.
- Air Force Manual (AFMAN) 63-119, *Certification of Readiness for Dedicated*

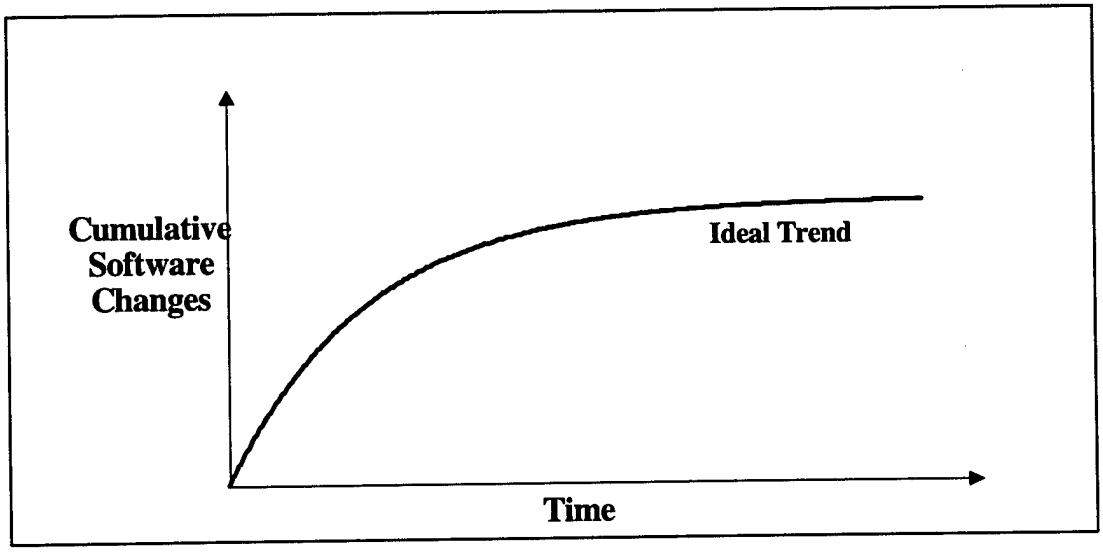


Figure 1.1. Ideal Software Change Rate.

Operational Test and Evaluation, presents templates that include the requirement that systems will not proceed to dedicated OT&E with any Priority 1 or 2 software problems that will affect an OT&E critical operational issue and the rate of problem identification must be decreasing.

- Department of Defense Operational Test and Evaluation Policy Letter, *Software Maturity Criteria for Dedicated Operational Test and Evaluation of Software-Intensive Systems* (31 May 94), states that systems will not proceed to dedicated OT&E with any Priority 1 or 2 software problems that will affect an OT&E critical operational issue.

1.4. Evaluation Planning.

1.4.1. **What are the primary responsibilities of the STM and DSE?** The STM will plan and conduct the evaluations and report the results for programs without an assigned DSE. Once assigned, the DSE will participate in the collection and scoring of software failures, conduct maturity evaluations, and report the results.

1.4.2. **Maturity Factors.** Maturity evaluations are based on various trends revolving around the identification and implementation of technical solutions to requested changes and problems. It is the responsibility of the STM/DSE to aggregate these trends and evaluate the maturity of the software and its potential impact on the system's ability to undergo the rigors of OT&E. Many outside factors might influence these trends. Paragraph 3.4 of this guide describes some common considerations (test rate, requirements stability, and test completeness) and their impacts on maturity evaluation. It is the responsibility of the DSE, in conjunction with the STM, to identify these external factors and account for their impact on the evaluation.

1.4.3. **Software Change Process.** Since every software development is unique, the number of different processes for collection of changes, problems, and the means for tracking corrections are vast. Instead of attempting to describe every situation, this pamphlet outlines typical procedures for large, medium-sized, and small software development programs. Regardless of size, software change requests require modification of one or more of the products illustrated in figure 1.2. Typically, requirements changes dominate the early portions of a software effort. As the program evolves, requested changes shift to design, source code, and test procedures. No matter which phase the software development is in, there should be maturity data that can be evaluated.

1.4.3.1. **Large Software Development Programs.** In a large acquisition program, the trouble-reporting and change-request process may be complex. One possible example is identified in figure 1.3. In this situation, software change requests are the result of either investigated problems, user requirements changes, adaptive hardware changes, or documentation changes. The evaluator must ensure the data used for a software maturity evaluation include failures, problems, and additional required changes. At the same time, it is important to verify that there is no duplication of data.

1.4.3.2. **Medium Software Development Programs.** In some programs, evaluators will find their

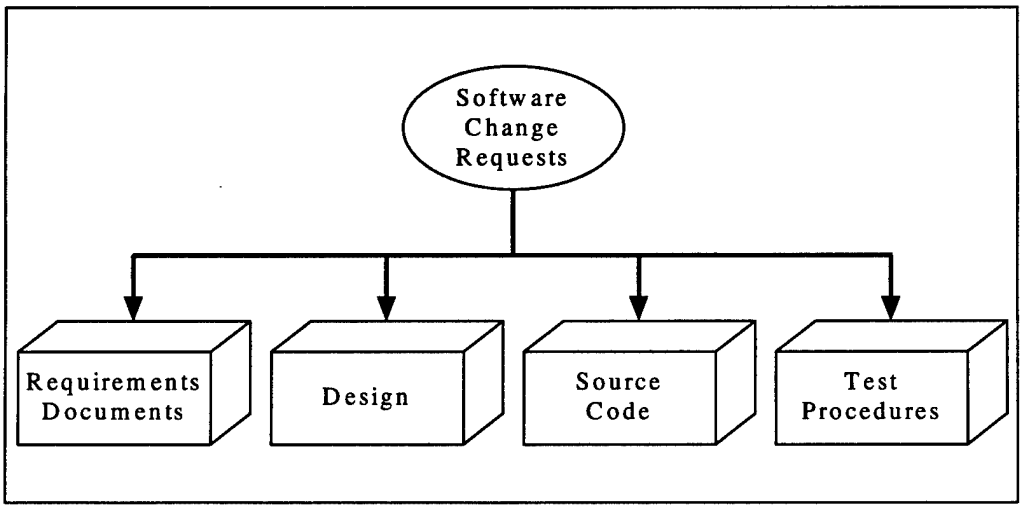


Figure 1.2. Types of Software Baseline Changes.

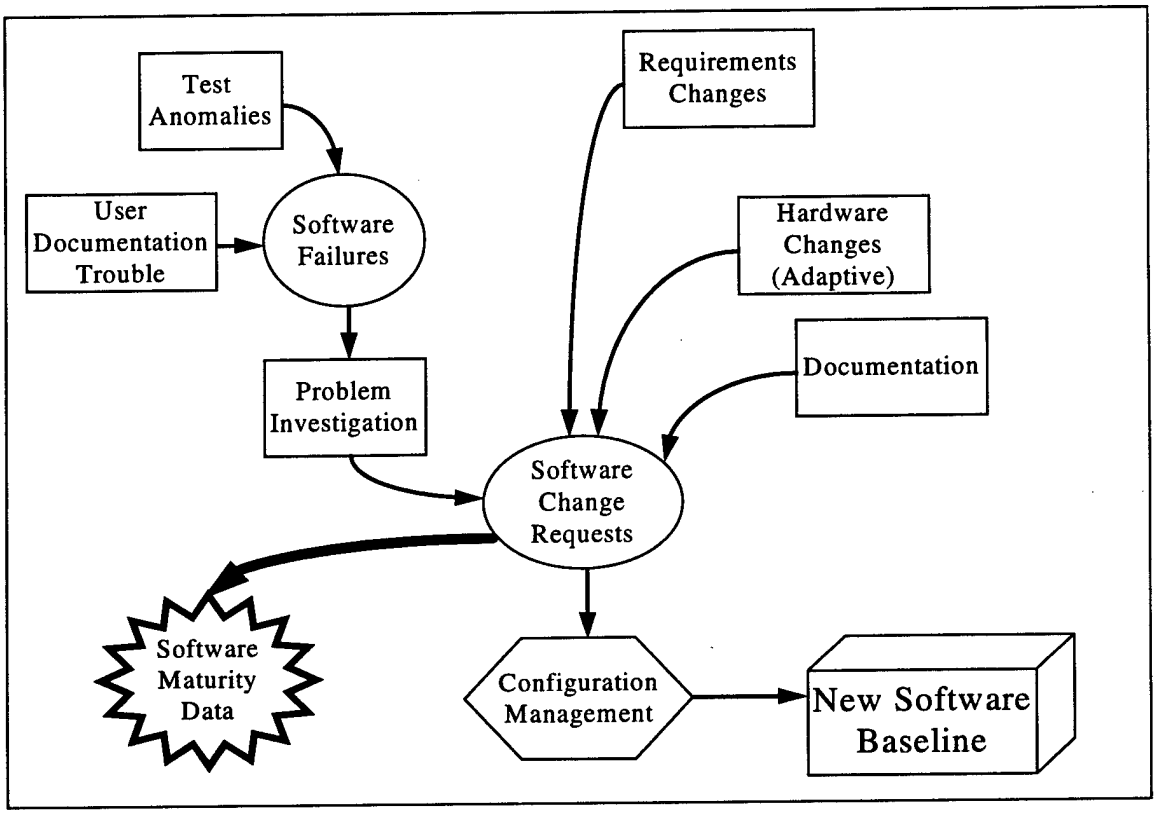


Figure 1.3. Typical Large-Program Software Change Process.

program has two unconnected databases; one that tracks software problems and another that lists new user requirements. Both sources of data can be used together to get a complete software maturity picture and can be used individually to focus on the source of maturity problems.

1.4.3.3. **Small Software Development Programs.** Most small software efforts track software problems and changes together in one process. This situation simplifies work for the STM since the data source is obvious and less likely to contain duplication.

1.4.4. **Test Planning Documentation.** During test planning, the STM must ensure software maturity is addressed in the appropriate acquisition and test documents. Arrangements must be made via the applicable Data Item Description (DID) on the Contract Data Requirements List (CDRL) to collect and report software change data. The major documents and recommendations follow.

- **Test Resource Plan (TRP).** Ensure the TRP identifies the necessary resources to collect/analyze software change data. It might be appropriate to have a dedicated evaluator to perform these duties in some of the larger tests. If so, identify this in the TRP.
- **Operational Requirements Document (ORD) and Requirements Correlation Matrix (RCM).** When reviewing/commenting on an ORD/RCM, ensure any specified software maturity requirements are consistent with the test readiness templates (AFMAN 63-119, *Certification of Readiness for Dedicated Operational Test and Evaluation*).
- **Test and Evaluation Master Plan (TEMP).** AFOTEC Instruction (AFOTECI) 99-101, *Management of Operational Test and Evaluation*, states the measures of performance and MOEs must address system maturity. Obviously software maturity is an important aspect of system maturity in most modern systems. Although TEMPs vary in format, attachment 1 presents an example software maturity MOE.
- **Test Plan.** Refer to AFOTECI 99-101 for details concerning the format of the test plan. Each test plan is unique and it is difficult to show how maturity should be addressed in a particular plan.

Chapter 2

SOFTWARE MATURITY DATA

2.1. Software Failure and Change Data. Software maturity evaluation depends on the adequate collection of change and failure data. We know all software systems have an unknown number of faults when delivered for test or use. In a broad sense, our goal is to measure the developer's progress toward finding and correcting these existing problems, but we're also interested in requirements stability, suggested enhancements, and design changes that may indicate the system is not ready for operational use. To evaluate this progress, we must collect and analyze change data. Attachment 2 lists all data required to evaluate software maturity.

...all software systems have an unknown number of faults when delivered for test or use...our goal is to measure the developer's progress toward finding and correcting these existing problems...

2.1.1. **Types of Changes.** As previously mentioned, changes can be grouped into adaptive, perfective and corrective changes. The term software change data are often used interchangeably with software problem data and software trouble data. Failure data are special cases of corrective change which becomes predominant during OT&E. For the purpose of our evaluations, AFOTEC uses the following definition of a software failure.

A *SOFTWARE FAILURE* is defined as the inability of a system's software component to perform a required function, as perceived by the user, within specified limits.

2.1.2. **Software Failure Process.** When a system failure occurs it can be traced to a fault or faults. A software fault, sometimes called a bug, is a software condition that causes a functional unit to fail to perform its required function. A fault or bug is caused by an error or defect that occurs during the software development effort. While software reliability is only concerned with these demonstrated failures and remaining faults, maturity includes all changes.

2.1.3. **Unintended Software Effects.** Whenever the topic of software failures is brought up, someone inevitably points out software does not break. Technically speaking, they are correct. Software does exactly what it is programmed to do. However, software can cause a system to fail, operate in a degraded mode, or not properly support the user in accomplishing the mission. The AFOTEC-adopted definition of software failure focuses on the system impact of a software failure. A dramatic example of software causing a system to fail occurred during Operation Desert Storm when an Iraqi Scud Missile hit an American barracks in Dhahran, Saudi Arabia. Twenty-eight Americans lost their lives. The United States Army determined the Patriot anti-missile defensive system did not fire at the incoming Scud because of a software failure in the computer system that tracks incoming missiles (Lauren Ruth Wiener. *Digital Woes*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1993).

AFOTEC-adopted definition of software failure focuses on the system impact of a software failure

2.2. Software Change Severity Levels. Software problems have different levels of severity. For example, a misspelled word on a menu may be assigned a low severity level. A software failure that causes the radar on an airborne warning and control system (AWACS) aircraft to shutdown (prohibiting mission accomplishment) would be assigned a high severity level. AFOTEC severity levels (refer to attachment 3) are basically the same as those found in the following publications:

- Department of Defense-Standard (DOD-STD) 2167A, *Defense System Software Development*, Appendix C (Category and Problem Classifications for Problem Reporting)
- Military-Standard (MIL-STD) 498, *Software Development and Documentation*, Appendix C (Category and Priority Classifications for Problem Reporting)
- AFI 10-602, *Determining Logistics Support and Readiness Requirements*, attachment 10 (Software Design and Supportability Measures)

2.3. Software Change Data Collection. Figure 2.1 shows the system acquisition life cycle. The chart also displays when the procedures for collecting

change data should be in place, when maturity data are actually collected, and finally when maturity evaluations take place. As explained earlier, different types of software maturity data are available at different stages of the development. During software requirements definition and allocation, changes to the requirements can be used to measure requirements stability. During the design phase, additional information about design changes can be added to on-going requirements change data collection. Each phase of the development adds additional information to the previous maturity data. The solid line for software maturity data collection (figure 2.1) indicates that changes and problems found during integration and testing are the primary source of software maturity data as a program moves toward OT&E. Note that data must be delivered on a regular basis to support planned evaluations as described in section 3.

2.3.1. **Data Collection and Delivery.** Depending on the evaluation schedule, data collection may begin as early as the beginning of the software requirements analysis phase of the program. If early evaluation of requirements maturity is not required, software maturity data collection must begin no later than when the software is placed under formal configuration control or at the beginning of system integration testing.

2.3.2. **Normal Data Collection Process.** Collection of software maturity data is almost always under way in a software development prior to AFOTEC involvement. Typically the program office places one of the following DIDs on contract with the software developer (depending on the standard used for a particular system):

- DOD-STD-2167A, *Defense System Software Development.*
  DI-MCCR-80030 - Software Development Plan
- MIL-STD-498, *Software Development and Documentation.*
  DI-IPSC-81427 - Software Development Plan

Delivery of this data should be reflected on the CDRL. As stated earlier, the SAF/AQ Policy Letter 93M-017, *Software Metrics,* directs system program offices (SPO) to collect various software metrics. One of the metrics, software quality, is very similar to software maturity. If the SPO is collecting the data to support software quality, AFOTEC can use that same data set to support software maturity evaluations. Although each program is different, the software maturity database is often identified as the software trouble report, software problem report, software change report, or test discrepancy report database.

In the rare case where this information is either not available or deliverable, the STM should identify and correct this situation early in the OT&E planning phase.

2.3.3. **Requesting Software Maturity Data.** The STM/DSE should send a letter as early as possible requesting the SPO task the developer to collect and

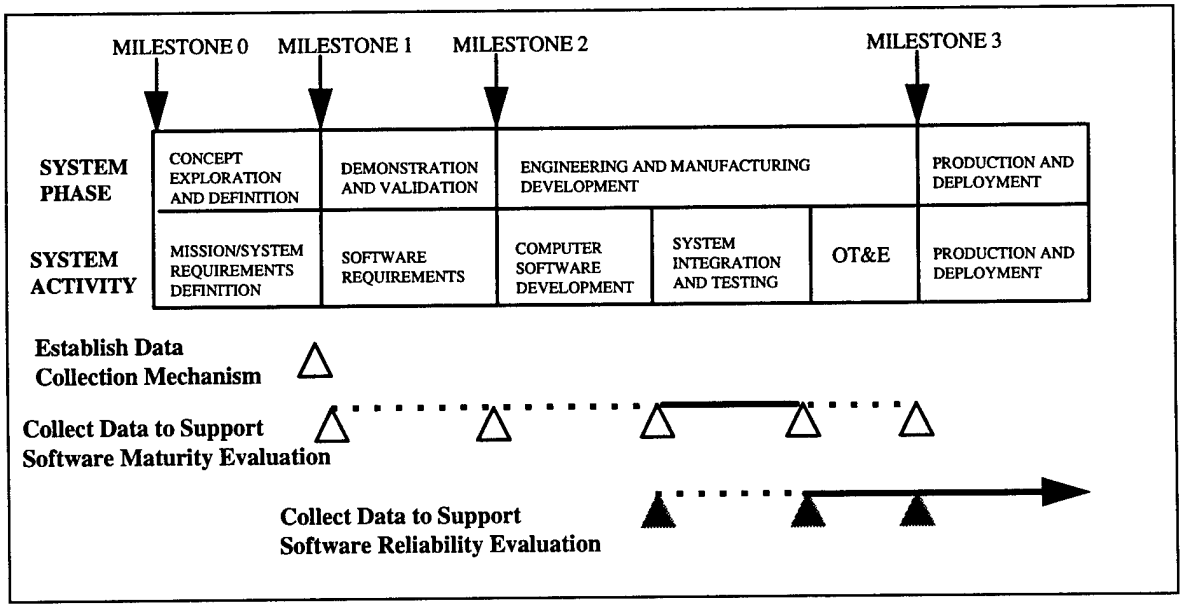


Figure 2.1. Software Change Data Collection.

send the data to the SPO and make arrangements for AFOTEC to receive the data. A sample letter of request is at attachment 4. The table at attachment 2 describes the required data and suggests additional data that could be collected for a more in-depth maturity evaluation. The maturity data should be delivered in electronic format (e.g., database, spreadsheet, or delimited text file) whenever possible.

2.3.4. **Data Integrity**. It is important for all interested organizations to agree on the software change request database format, scoring, and responsibility for upkeep of the database. Typically, the developer is responsible for assigning severity levels and verifying uniqueness of change requests during integration testing. Later, during developmental test and evaluation, the program office or test team may also be involved in scoring SPRs and change requests. During OT&E, the STM/DSE is responsible for tracking software problems and failures. The Joint Reliability and Maintainability Evaluation Team (JRMET) verifies and scores each problem and failure during operational testing while the Test Data Scoring Board is responsible for resolving scoring conflicts as a government only board. As a member of these groups, the STM/DSE will provide major input to scoring and evaluating software-related problems during OT&E.

Chapter 3

MATURITY EVALUATION AND REPORTING

3.1. Software Maturity Evaluation.

3.1.1. **Timing**. AFOTEC can begin maturity analysis as soon as data are collected and delivered. Depending on the system's complexity, size, or oversight, the STM/DSE will decide how often to conduct and report maturity evaluations. Evaluations are normally conducted more often as OT&E approaches. For many systems, maturity is evaluated monthly and reported quarterly, but more frequent evaluations may be necessary prior to the operational test readiness certification. The evaluator should also consider additional maturity evaluations and reports to coincide with acquisition events. For example, maturity could be evaluated regularly, and reported for program management reviews, test readiness reviews, or other decision events.

For many systems, maturity is evaluated monthly and reported quarterly, but more frequent evaluations may be necessary prior to the operational test readiness certification.

3.1.2. **Workload**. This evaluation should not be a time burden on the evaluator. It is intended to require no more than 2 days to perform a maturity evaluation and report results. If your evaluation requires significantly more time, contact HQ AFOTEC/SAS for help in reducing the effort required.

3.1.3. **Indenture Level**. For most programs, software maturity evaluations should be conducted at the computer software configuration item (CSCI) level as well as the system level. Conducting the evaluation on each CSCI is important because sometimes good software hides the high failure rate of other software, or the reverse might also be true. In addition, evaluating maturity at the CSCI level helps the SPO more effectively address problems with immature software components. Determining what indenture level is appropriate for your system is a function of three factors.

3.1.3.1. **Length of Time Software Maturity Data are Collected**. Most trends require a minimum of 10 to 15 time periods to clearly demonstrate trends. These time periods could be weeks, months, quarters, or years. It will be difficult to evaluate maturity at any indenture level without at least 10 weeks of maturity data.

3.1.3.2. **Number of Changes**. Although the number of changes required to show trends is related to the length of time the data are collected, a minimum of about 50 changes is a good rule of thumb. If a CSCI does not have approximately 50 changes, it may be better to evaluate maturity at the software system level, rather than at the CSCI level.

3.1.3.3. **New or Modified Lines of Code**. Finally, if a CSCI consists of less than 30,000 new or modified lines of code, the defect density will be extremely volatile at the CSCI level. This means that a single remaining change will have an unusually large impact on the defect density of the CSCI.

3.1.3.4. **Large Programs**. Extensive data collection efforts within some programs enable the software

For most programs, software maturity evaluations should be conducted at the CSCI level as well as the system level.

evaluator to further focus the maturity evaluation. If software maturity data can be tracked to the computer software component (CSC) or computer software unit (CSU), very specific problem areas can be identified. Since most CSCs or CSUs will be fewer than 30,000 new or modified lines of code, defect density information is best evaluated at the CSCI level.

3.1.3.5. **Small Programs**. Many small programs do not have the number of problems, length of data collection, or number of new or modified lines of code required to create trends at the CSCI level. For these small programs, a meaningful maturity evaluation can only be accomplished at the system level.

3.1.4. **Synthesis of Multiple Trends**. It is important to keep in mind no single trend is a direct measure of software maturity. All trends must be considered together. Figure 3.1 emphasizes this point. Additional trends beyond the ones covered here may be appropriate for a particular system. The test team, DSE, and STM should agree on what trends to evaluate, how to collect the data, and how to analyze the trends.

It is important to keep in mind no single trend is a direct measure of software maturity. All trends must be considered together.

3.1.5. **Reporting**. Maturity should be reported to the lowest reasonable level. While small programs may only have meaningful results at the system level, most programs should also include results at the CSCI level. It is the responsibility of the STM/DSE to investigate the underlying causes of the indicators and trends. Attachment 5 presents a high-level outline of a typical software maturity report.

3.1.6. **Archival of Results**. As with other AFOTEC software evaluations, provide your HQ AFOTEC/ SAS point of contact with a copy of your evaluation data, charts, and report. These items are an important part of our historical data used to improve current methodologies and research new evaluation techniques.

3.2. Weighting. Some of the maturity trends are "weighted" based on the severity level of the failure. The weighting factor is listed in table 3.1. The failures/changes are multiplied by their respective weighting factor to produce values called "change points." These change points are used to produce some of the trends. Each trend will be identified as weighted or unweighted.

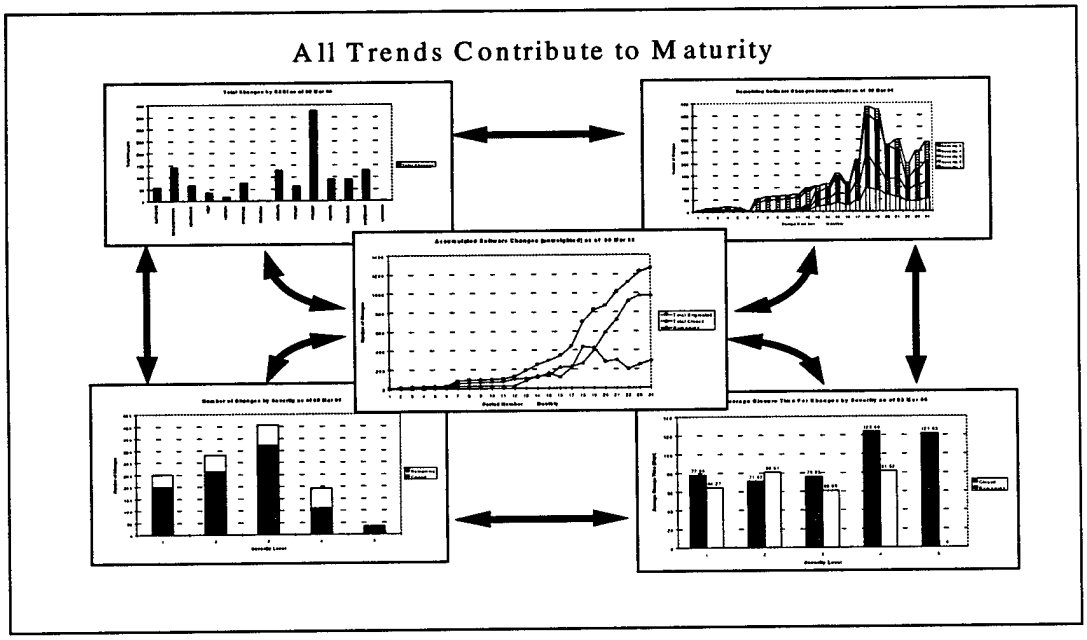


Figure 3.1. Software Maturity is a Synthesis of Many Trends.

Table 3.1. Software Severity Levels and Weighting Factors.

Severity Level	Title	Weighting
1	System Abort	30
2	System Degraded - No Work-around	15
3	System Degraded - Work-around	5
4	System Not Degraded	2
5	Minor Change	1

3.3. Sample Charts. Attachment 5 presents sample maturity charts and explains the analysis of each chart. All charts in attachment 5 were produced by the HQ AFOTEC/SAS Maturity Evaluation and Analysis Tool (MEAT) version 3.0. This evaluation guide may be used with MEAT version 3.0 or higher.

3.4. Other Considerations. Software maturity trends cannot be fully understood without specific program knowledge described in the following paragraphs. The charts presented in this section are intended only as examples and are not produced by MEAT version 3.0. The goal of this section is for the evaluator to understand the impact of the external factors on maturity evaluation.

3.4.1. **Test Rate.** The software maturity trends must be used in conjunction with data representing test progress and completeness. The rate of testing is required because changes in this rate will affect the slope of the *total originated changes* curves in the weighted and unweighted accumulated software change charts. If testing slows, the slope of the changes (problems) discovered curve should decrease since fewer failures should be found per unit time. Conversely, if the rate of testing increases, the number of changes (problems) found per unit time potentially increases, thereby causing a steeper slope in the changes (problems) discovered curve. The data required to illustrate test rate is included in attachment 2. Figure 3.2 is a sample test rate chart. It is not produced by MEAT version 3.0. Since many program offices already collect and present this information, most evaluators should simply obtain current test rate information from the program office.

3.4.2. **Test Completeness.** This measure helps the evaluator to determine how many of the formally identified test procedures have been successfully accomplished. This understanding helps to estimate confidence in the overall maturity evaluation. Test completeness is expressed as a ratio between the number of successfully passed test procedures and the total number of test procedures. In many cases, these data are readily available from the SPO (see

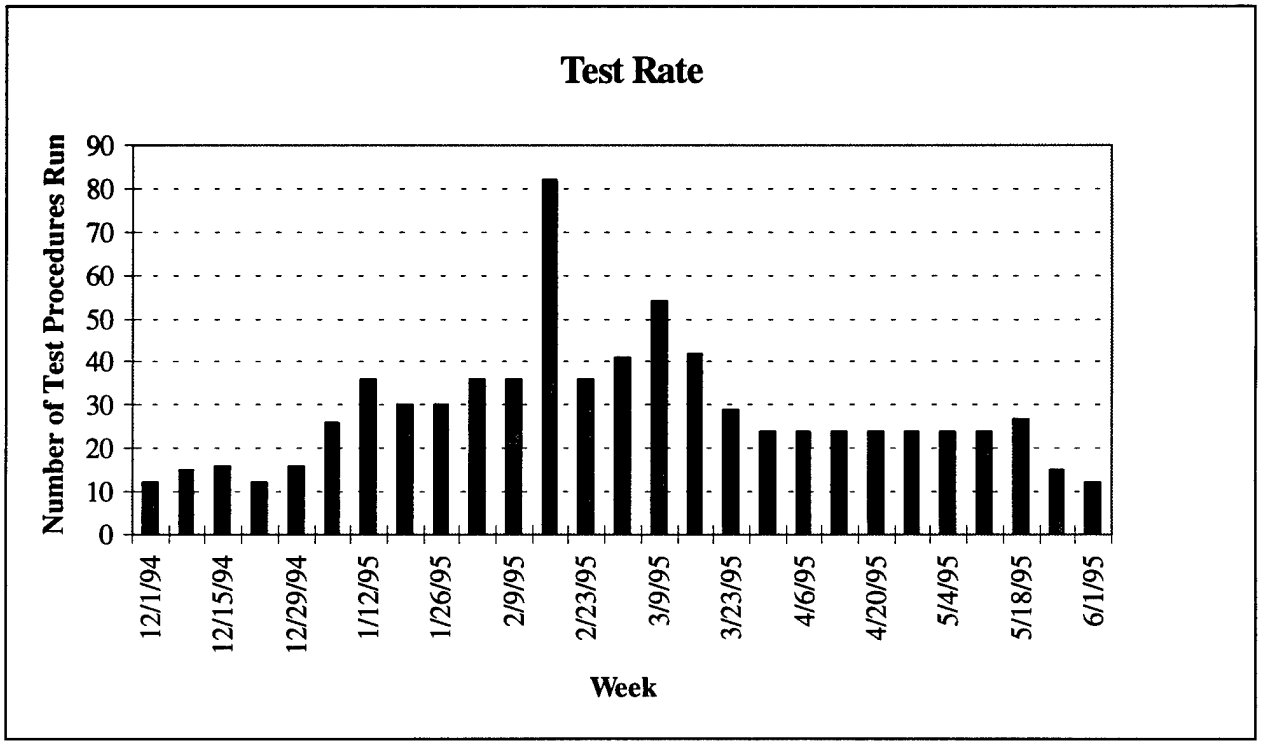


Figure 3.2. Test Rate by Week.

attachment 2). Figure 3.3 is a sample test completeness chart. Notice that the total number of test procedures normally grows as the program evolves. The evaluator must remember that successfully completing all test cases (complete testing) does not guarantee thorough testing. Traceability between test procedures and requirements or functions, which is not part of this chart, is necessary to verify thorough functional testing.

Traceability between test procedures and requirements or functions, which is not part of this chart, is necessary to verify thorough functional testing.

3.4.3. **Requirements Stability.** One of the difficulties in determining the status of software intensive systems is software requirements are a moving target. This occurs for several reasons including:

- Initial user requirements are inadequately defined.
- User requested changes after the initial requirements baseline are not adequately controlled.
- Requirements originally assigned to hardware are subsequently reassigned to software.

In addition to a changing environment, the SPO may have difficulty translating the user requirements into contract specifications. As stated earlier, a software maturity evaluation is concerned with all types of software changes. A good understanding of the nature of software changes within a system can help the evaluator determine the root cause for software maturity problems. Data about the type of change (perfective, adaptive, or corrective) can be used to measure requirements stability (see attachment 2).

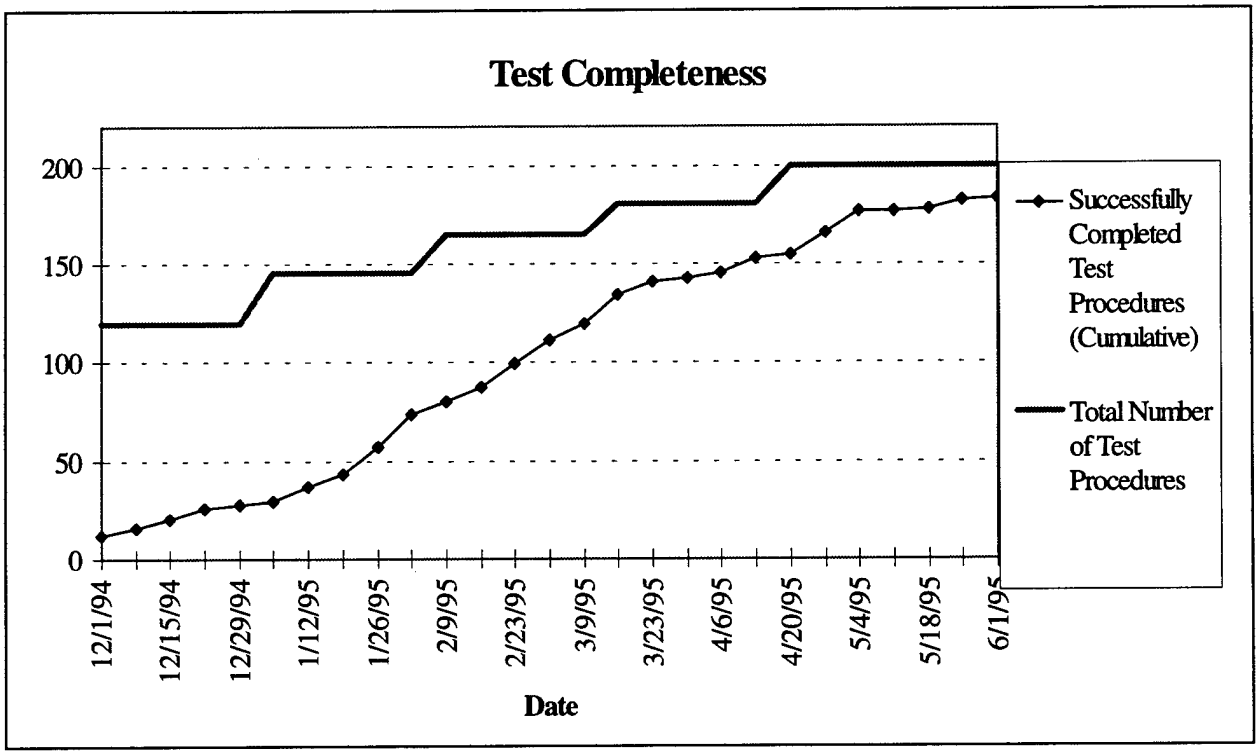


Figure 3.3. Test Completeness.

Chapter 4

Lessons Learned

4.1. Section Overview. This section is designed to be a collection of experiences and guidance. It addresses common problems found by STMs/DSEs and typical solutions to those problems. Each suggested action should be applied with common sense to a specific program. From their experiences, evaluators are encouraged to provide additional problems and solutions to improve this section. The notes section can be used to write comments about the applicability of these problems and suggested actions to your specific programs.

4.2. No DSE.

4.2.1. **Problem**: No DSE is on the test team

4.2.2. **Suggested Action**: The STM will perform all of the DSE duties.

4.3. Not Ready for Test.

4.3.1. **Problem**: Pre-OT&E evaluations indicate the software is not ready for OT&E but it starts anyway.

4.3.2. **Suggested Action**: The test readiness templates should enable the evaluator to highlight this during the test readiness certification process. If the choice is made to proceed to OT&E despite this risk, the evaluator should continue the maturity evaluations into OT&E. The final report should reflect the state of software maturity at the end of OT&E and should also state that software maturity was deemed inadequate at the start of test (i.e., a known system deficiency at the start of OT&E).

Notes:

4.4. Incremental or Evolutionary System

4.4.1. **Problem**: The system is being acquired as an incremental or evolutionary acquisition.

4.4.2. **Suggested Action**: Perform the maturity evaluation on each release scheduled for OT&E

4.5. Duplicate Items or Deleted Items in Software Maturity Database.

4.5.1. **Problem**: Duplicate or canceled software problems cause the maturity analysis to be incorrect.

4.5.2 **Suggested Action**: Ensure duplicate, disapproved, or canceled software change requests are deleted from the database prior to analysis. The following two items are examples of exceptions to this rule:

- A change request that was disapproved only because it was out of contract scope is still a valid request and should remain in the database.
- Identified changes which are "on hold" or awaiting future releases should remain in the database

4.6. Nonstandard Severity Level Definitions.

4.6.1. **Problem**: Developer or program office uses different severity-level definitions or uses a priority system.

4.6.2. **Suggested Action**: The evaluator should make an attempt to map the priority or severity levels into the standard five-level format. In past cases, systems that use only three severity levels have been successfully mapped to 1, 3, and 5 in the standard severity-level format. This enables the evaluator to use the weighted maturity charts. If the severity levels cannot be mapped to the standard five-level format, the weighted maturity charts will be meaningless. In these cases, the evaluator should focus on the unweighted charts and completely disregard the weighted charts.

Notes:

GEORGE B. HARRISON
Major General, USAF
Commander

SAMPLE SOFTWARE MATURITY MOE

A1.1. MOE X-Y. Software Maturity. This evaluation measures the system software's progress toward meeting documented user needs. The developer, development test, and operational test team will collect software change data in order to track the system's ability to meet requirements. As the development progresses, fewer major problems should be found. This trend will indicate whether the system is maturing toward meeting its operational requirements.

A1.2. *Evaluation Criteria*: If no ORD software maturity requirements exist, then evaluation criteria is in accordance with AFOTEC Pamphlet (AFOTECP) 99-102, volume 6, *Software Maturity Evaluation Guide*. Department of Defense (DoD) and AF/TE guidance on software maturity provide additional evaluation criteria.

SOFTWARE MATURITY DATA

A2.1. Mandatory data for basic software maturity trends.

Data Item	Format	Notes
Software Change/Problem Number	Character, Number, or Alphanumeric	Must be unique
CSCI	Character	Use a standard set of nomenclature or acronyms
Severity of Change/Problem	Number	Use definitions to assign severity from 1 to 5
Date Change Requested or Problem Discovered	Date	
Date Change Closed or Problem Closed	Date	
Description of Change/Problem	Text	

A2.2. Optional data required to evaluate change density and remaining change density.

Data Item	Format	Notes
Software Size	Number	New or modified source lines of code, function points, or other measure by CSCI, CSC, or CSU (use same measure for each portion)

A2.3. Optional data required for more thorough analysis of basic maturity trends.

Data Item	Format	Notes
Computer Software Component (CSC)	Character	The additional data can provide more specific insight into specific maturity problem areas
Computer Software Unit (CSU)	Character	The additional data can provide more specific insight into specific maturity problem areas
Type of Change (Adaptive, Perfective, or Corrective)	A, P, or C	Assist in determining whether maturity problems are due to requirements instability, development defects, or changes in the environment.
Category of Software Change	Text (e.g., requirements, design, code, data, test, or manuals)	Provides further insight into the source of software problems. (See definitions in MIL-STD-498, Appendix C.)

A2.4. Optional data required to evaluate test rate and test completeness.

Data Item	Format	Notes
Total Number of Test Procedures	Number	The together with the following two data items allow the evaluator to estimate test completeness
Total Number of Test Procedures Exercised	Number	
Total Number of Test Procedures Passed	Number	
Test Procedure/Run Date	Test Procedure Identifier and Run Date	The data can be used to describe test rate

SOFTWARE CHANGE SEVERITY LEVELS

A3.1. Severity Level 1 (System Abort).

A3.1.1. A software change is categorized with this severity level if one or more of the following impact statements apply:

A3.1.1.1. Prevents the accomplishment of an operational mission-essential capability.

A3.1.1.2. Prevents the operator's accomplishment of an operational or mission-essential capability.

A3.1.1.3. Jeopardizes safety.

A3.2 Severity Level 2 (System Degraded -- No Work-Around).

A3.2.1. A software change is categorized with this severity level if one or more of the following impact statements apply:

A3.2.1.1. Adversely affects the accomplishment of an operational or mission-essential capability for which no alternative work-around solution is known (program restarts/reboots are not acceptable work-around solutions).

A3.2.1.2. Adversely affects the operator's accomplishment of an operational or mission-essential capability for which no alternative work-around solution is known (Program restarts/reboots are not acceptable work-around solutions).

A3.3. Severity Level 3 (System Degraded - Work-Around).

A3.3.1. A software change is categorized with this severity level if one or more of the following impact statements apply:

A3.3.1.1. Adversely affects the accomplishment of an operational or mission-essential capability but a work-around solution is known.

A3.3.1.2. Adversely affects the operator's accomplishment of an operational or mission-essential capability but a work-around solution is known.

A3.4. Severity Level 4 (System Not Degraded).

A3.4.1. A software change is categorized with this severity level if the following impact statement applies:

A3.4.2. Results in user/operator inconvenience or annoyance but does not degrade a required operational or mission-essential capability.

A3.5. Severity Level 5 (Minor Change).

A3.5.1 Any other change is classified as severity level 5. Many documentation changes are considered severity level 5.

SAMPLE SOFTWARE MATURITY DATA REQUEST LETTER

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE OPERATIONAL TEST AND EVALUATION CENTER
KIRTLAND AIR FORCE BASE, NEW MEXICO

Date

MEMORANDUM FOR _____ SPO

FROM: HQ AFOTEC/SAS
8500 Gibson Blvd SE
Kirtland AFB NM 87117-5558

SUBJECT: Software Maturity Data Requirements/Analysis for the _____ Program

1. HQ AFOTEC is tasked by Program Management Directive to perform an Initial Operational Test and Evaluation (IOT&E) of _____ program. Software evaluations are an integral part of the overall IOT&E effort. Currently, AFOTEC is using a software maturity metric to aid decision-makers in determining the status of software intensive systems by providing an indication of the development progress of the software. Software maturity is a measure of the software's progress in its evolution toward satisfying all documented user requirements. The primary indicator of this evolution is the trend in the number and severity of software changes and failures plotted over time.

2. To develop a meaningful trend, software problem and change data must be collected when configuration control and/or software integration begins. Therefore we request the information in attachment 1 be collected and provided to **HQ AFOTEC/SAS** on a monthly basis beginning _____. Attachment 2 defines the software change severity levels that should be assigned to each problem/change.

3. The data required to perform the analysis should be readily available by the software development contractor and this data collection should not inhibit the contractor's daily activities. We appreciate your cooperation in this matter. If you have any questions or would like further information about software maturity, please contact _____.

Signature Block

Attachments
1. Software Maturity Data (all data from AFOTECP 99-102, vol 6, attachment 2)
2. Software Severity Levels

Figure A4.1. Sample Software Maturity Data Request Letter.

SAMPLE SOFTWARE MATURITY EVALUATION REPORT OUTLINE

I. **Executive Summary**

Identify the evaluator, software program, evaluation methodology, the software maturity database date, and the evaluation results. For some reports, it may be appropriate to discuss the software maturity impact on operational test readiness and operational test success. (half page)

II. **Detailed Results**

1. **System-Level Software Maturity**

All programs must report this level of software maturity. For smaller programs, this may be the only reporting level. Each meaningful trend (chart), and other external factors should be presented and discussed. (one page or less for each chart and discussion)

2. **CSCI #X Maturity**

Most programs will report down to this level and will include sections for each CSCI. Each meaningful trend (chart) should be presented with a brief discussion. (one page or less for each chart and discussion

a. **CSC #Y Analysis**

Present important maturity information at the CSC level. Repeat for each CSC that has a significant impact on the maturity or immaturity of a particular CSCI. (one page or less for each chart and discussion

III. **Summary and Recommendations.**

Reemphasize software maturity problem areas, potential solutions, and impact on operational testing.

Figure A5.1. Sample Software Maturity Evaluation Report Outline.

Ideal and Sample Maturity Charts

A6.1. All of the figures in this attachment were generated by the HQ AFOTEC/SAS Maturity Evaluation and Analysis Tool (MEAT) version 3.0. Contact HQ AFOTEC/SAS for a copy of the tool, the MEAT user's guide, or for further information.

A6.2. Accumulated Software Changes (Weighted). This chart contains three curves, *total originated, total closed*, and *remaining weighted software changes*. Each curve is the accumulated change points versus time. Figure A6.1 is a sample chart from an actual program.

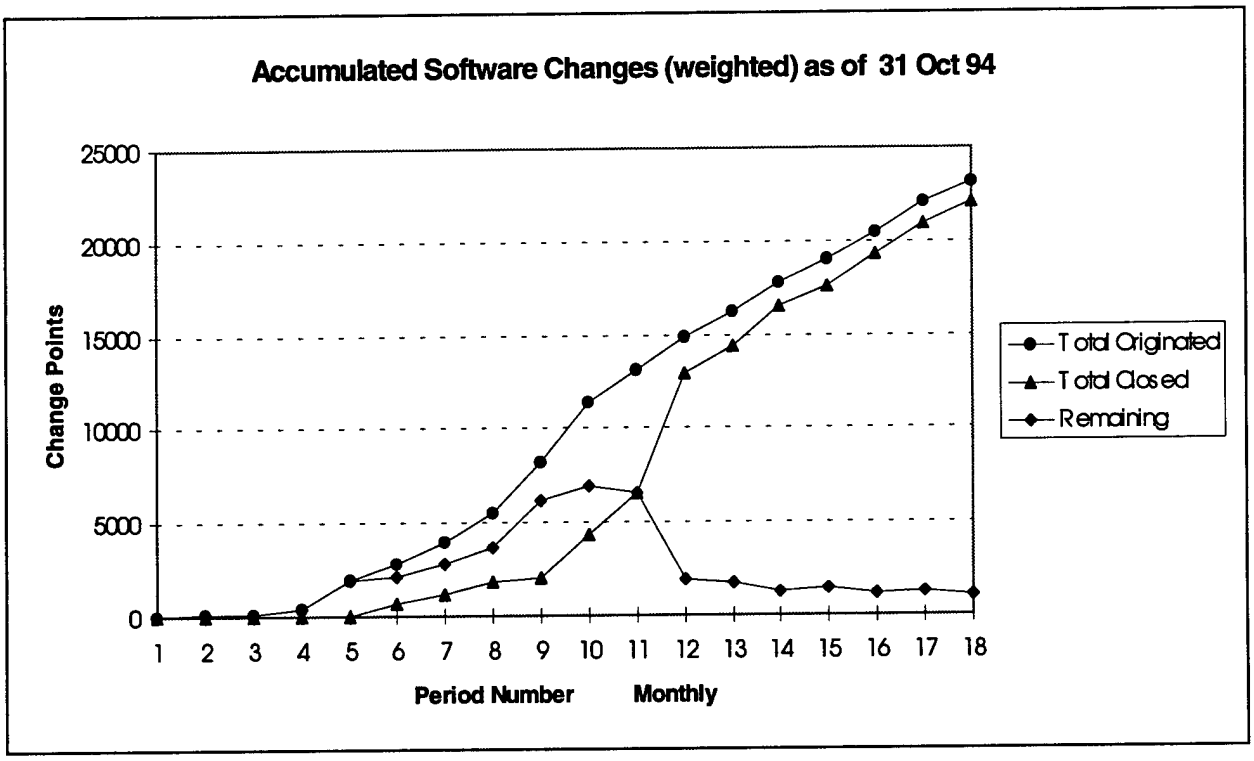


Figure A6.1. Accumulated Software Changes (Weighted).

A6.2.1. **What Are We Looking For?** The ideal *total originated* curve should begin to level off as the system approaches OT&E. In the above example, the developers/testers continue to find problems at a nearly constant rate. This is an indication the system is not mature. Figure A6.2 represents an ideal weighted chart. Compare the ideal chart to figure A6.1. The ideal *total closed* curve should follow the *total originated* curve and actually get closer to it as time passes. In figure A6.1, the developer is keeping up with changes, but just can't quite work the backlog of problems.

A6.2.2. The ideal *remaining* curve should become closer to zero as the system approaches OT&E. As mentioned in the previous paragraph, the system shown in figure A6.1 continues to have a consistent backlog of software changes.

CAUTION: As noted earlier, each trend must be considered in context of program schedule, test rate, test completeness, requirements stability, change density, and other external factors which affect software maturity. These factors can cause immature systems to appear mature.

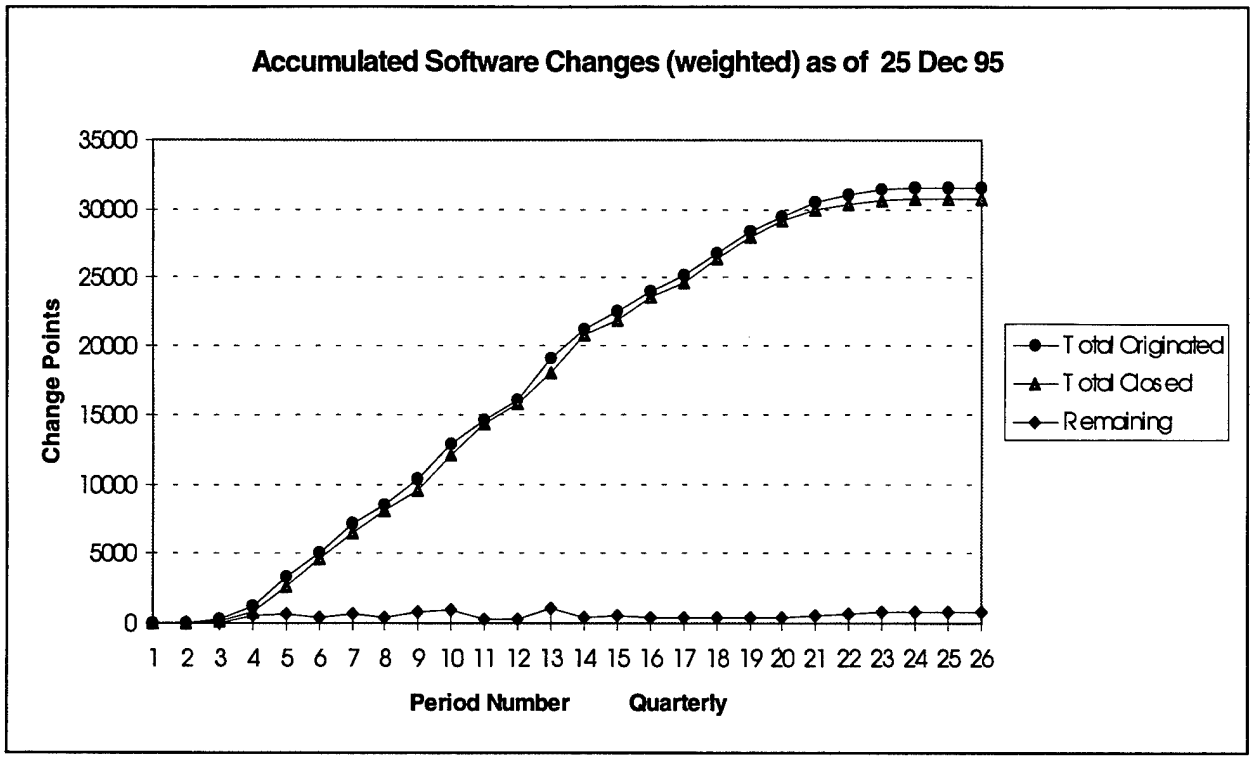


Figure A6.2. Ideal Accumulated Software Changes (Weighted).

A6.3. Accumulated Software Changes (Unweighted). This chart contains three curves, *total originated*, *total closed*, and *remaining software changes*. Each curve is the accumulated number of changes versus time. Figure A6.3 is an example chart.

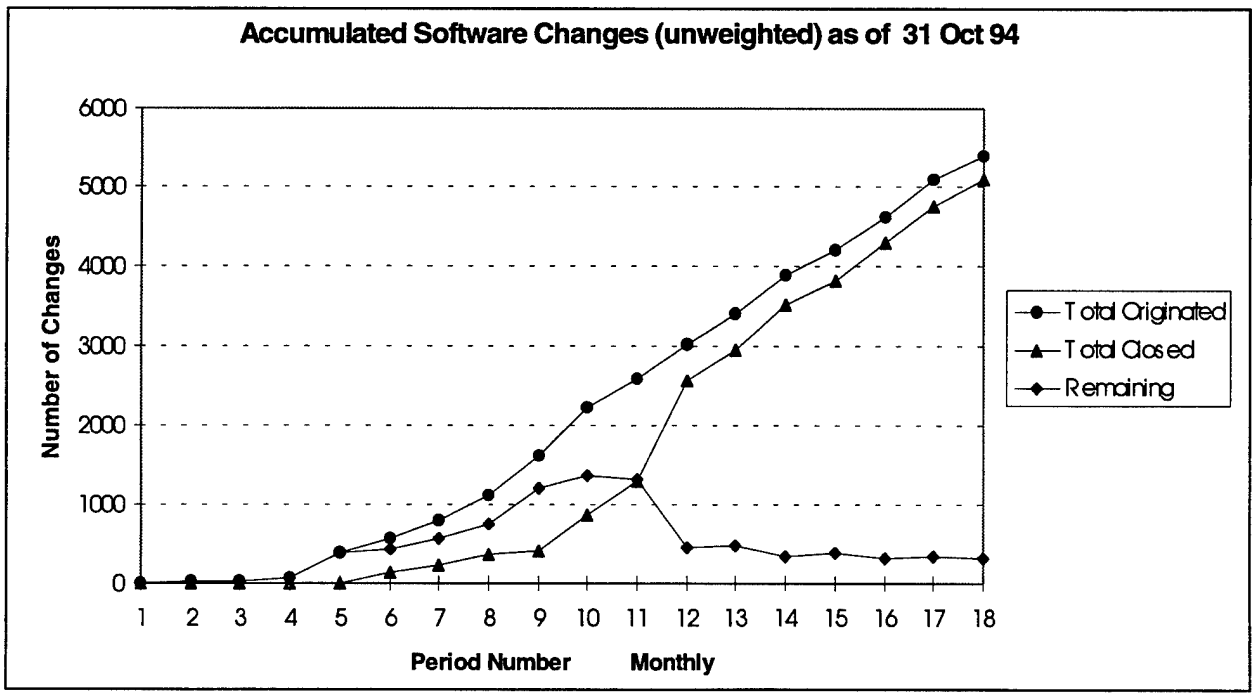


Figure A6.3. Accumulated Software Changes (Unweighted).

A6.3.1. **What Are We Looking For?** As described in the weighted version of this chart, the ideal total originated curve should begin to level off as the system approached OT&E. In the above example, the developers/testers continue to find problems at a consistent rate. This is an indication the system is not mature.

A6.3.2. The ideal *total closed* curve should follow the *total originated* curve and actually get closer to it as time passes. In this case, the developer is keeping up with changes, but just can't quite work the backlog of problems.

A6.3.3. The ideal *remaining* curve should become closer to zero as the system approaches OT&E. As mentioned in the previous paragraph, the system shown in figure A6.3 continues to have a consistent backlog of software changes.

A6.3.4. A sample ideal curve for the unweighted chart has the same shape as the ideal weighted chart shown in the previous section.

CAUTION: Each trend must be considered in context of program schedule, test rate, test completeness, requirements stability, change density, and other external factors which affect software maturity. These factors can cause immature systems to appear mature.

A6.3.5. **What is the difference between the weighted and unweighted charts**? For most programs, the unweighted chart will match the weighted chart, but figure A6.4 is a weighted version of the same data presented in figure A6.3. This chart was included to demonstrate how weighting can affect the evaluator's impression of maturity. If we only used an unweighted chart (figure A6.3), we would see a nearly constant rate of opening new changes. The weighted chart (figure A6.4) also shows that since month 15, the changes have been of increasingly higher severity levels. As you can see, we use both charts together to help complete the maturity picture.

- MEAT version 3.0 also provides similar charts for each individual severity level and for each CSCI.

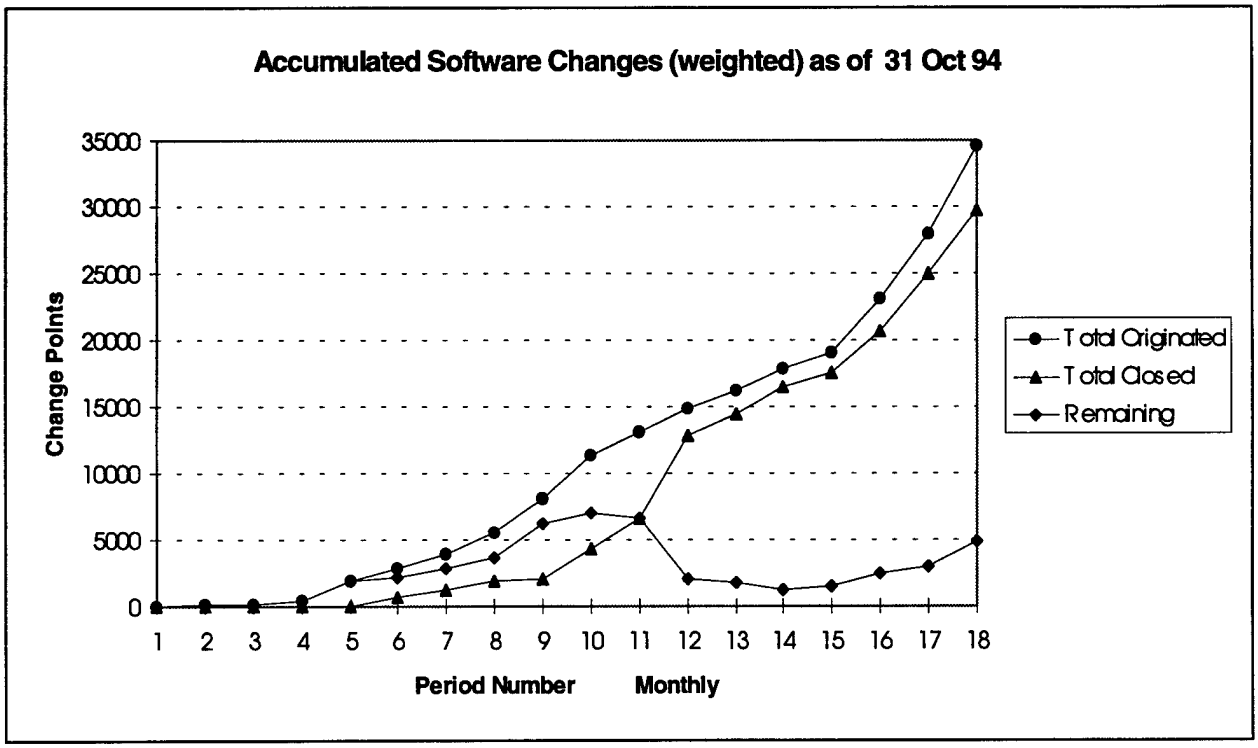


Figure A6.4. Accumulated Software Changes (Weighted).

A6.4. Average Severity of All Software Changes. This chart contains three curves, *total originated*, *total closed*, and *remaining software change* severity curves. Each curve is the average severity (change points) versus time. Figure A6.5 is a sample chart from an actual program.

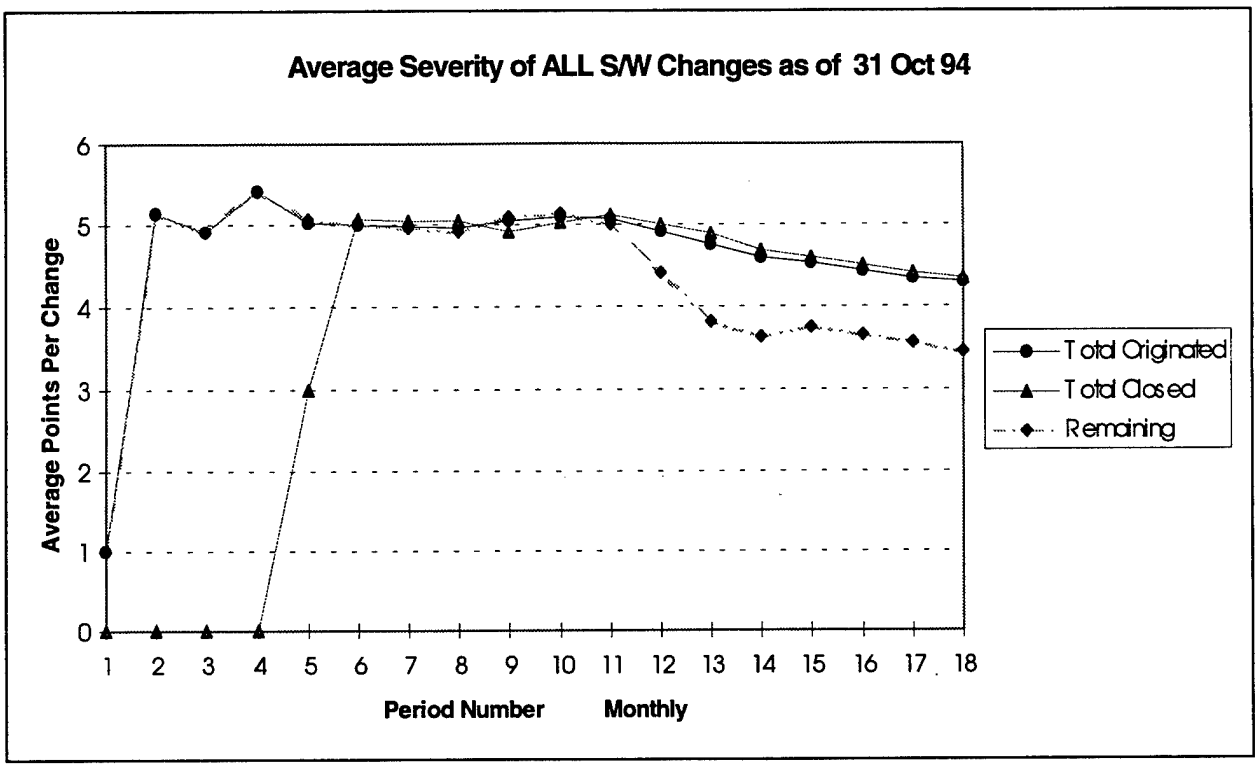


Figure A6.5. Average Severity of All Software Changes.

A6.4.1. **What Are We Looking For?** In the *total originated* curve, we hope to find a downward trend in average severity. This would indicate testing has found the most severe problems early and the system is capable of accomplishing its required mission with only minor bugs remaining. Even if this curve is decreasing, an evaluator should be concerned if the average severity is too high. In the above example, the average remaining severity hovered around five points per change until month 11. This equates to a severity level 3 problem (see table 1).

A6.4.2. Figure A6.6 is an ideal Average Severity Chart. Compare this chart to the sample shown above. The ideal *total closed* curve is above the *total originated* curve. This indicates the developer is fixing more critical problems before the lower priority changes. The ideal *remaining* curve is below the total *originated curve*. This indicates the remaining problems are of lesser impact than those that have already been closed.

A6.5. Average Closure Time For Changes by Severity. This chart consists of bar charts depicting the average closure time for closed changes and the average open time of unclosed changes for each severity level. Figure A6.7 is a sample chart from an actual program.

A6.5.1. **What Are We Looking For?** This chart can be used to get a **rough** estimate of how long a change of a given severity will take to implement. A developer's schedule estimate is based on program knowledge, size of change, and programmer workload. A developer's estimate is normally far more accurate for a specific change. The evaluator can also use this chart to estimate how much longer an open change will take to implement. Once again, this estimate is **rough**. Extremely high average closure times may also indicate that some problems are not being worked at all.

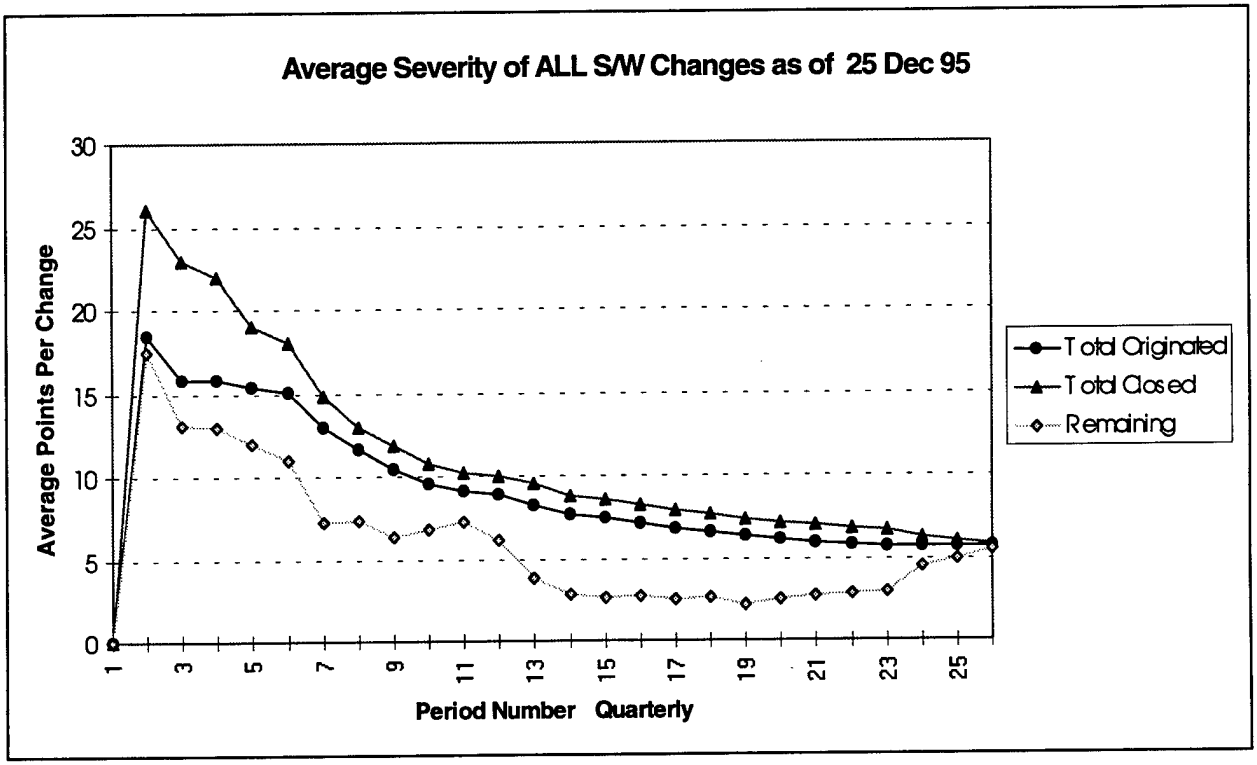


Figure A6.6. Ideal Average Severity Chart.

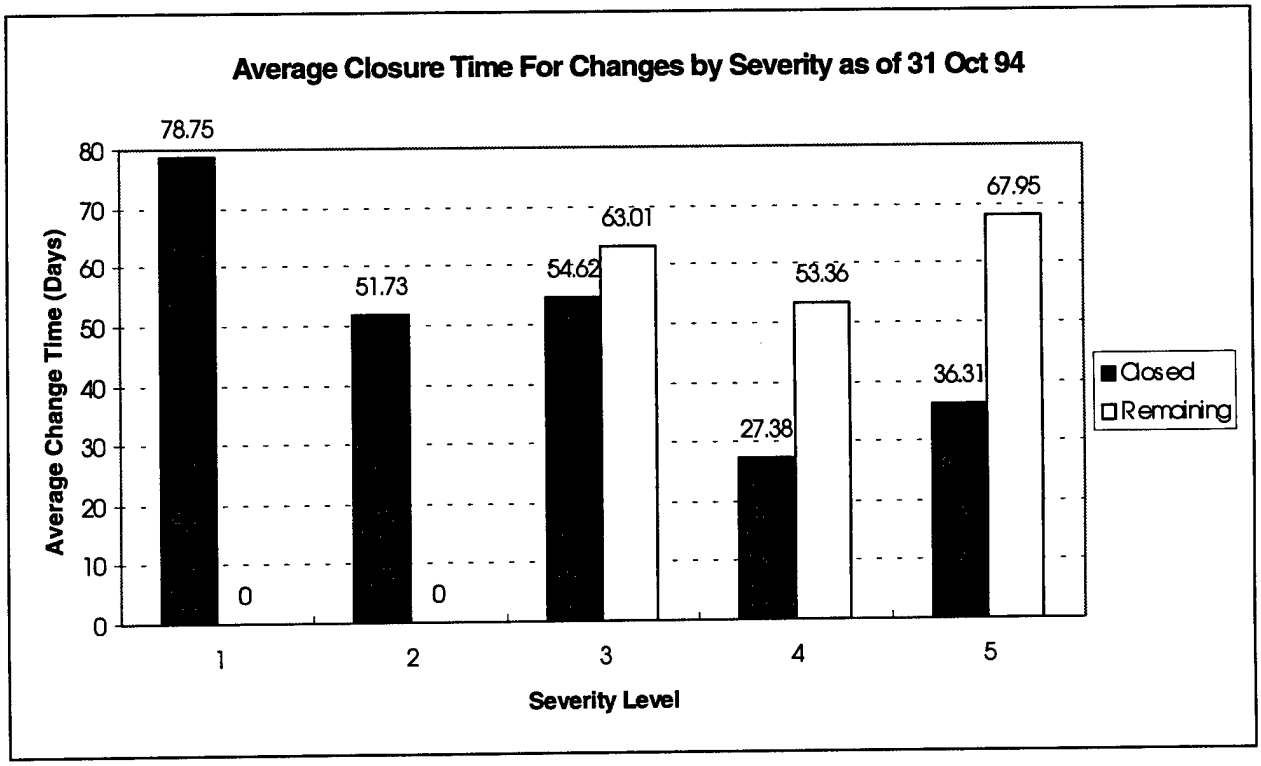


Figure A6.7. Average Closure Time For Changes by Severity.

A6.5.2. While these averages may hold up statistically, the closure time for a particular change is more closely tied to its difficulty than its severity level. It is possible to calculate confidence intervals for closure times from the raw maturity data.

Severity levels do not necessarily indicate the amount of effort required to implement a suggested change or identified problem.

A6.6. Number of Changes by Severity. This chart consists of bars depicting the number of changes for each severity level. The bars consist of closed (darkened) and open (white) portions. Figure A6.8 is a sample chart from an actual program.

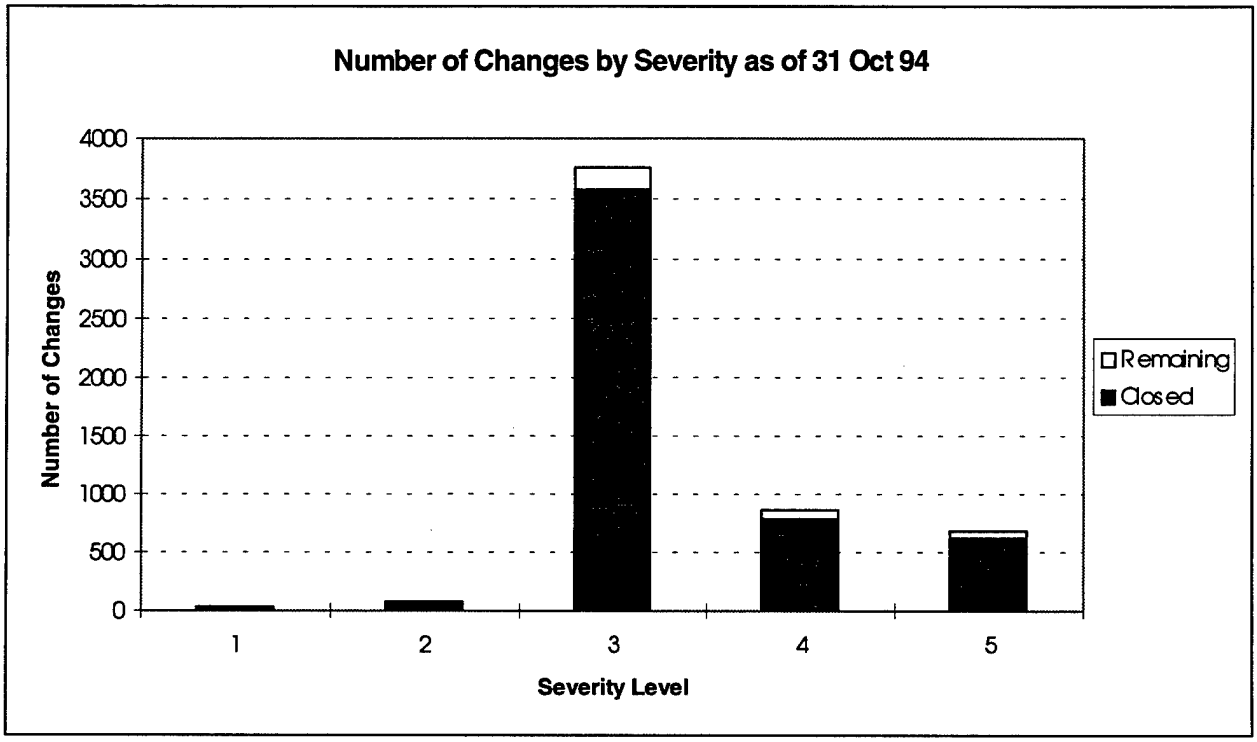


Figure A6.8. Number of Changes by Severity.

A6.6.1. **What Are We Looking For?** A well developed system should have relatively few severity 1 and 2 problems. Historically, most problems are found to be severity 3 and many low-severity problems are also normal. The sample chart exaggerates these norms with an unusually large number of severity 3 problems. Figure A6.9 is an example of the ideal severity level distribution. Notice that the ideal chart shows a system with no open severity level 1 or 2 problems.

HINT: This chart tends to make the number of remaining changes look small in comparison to the changes already closed. Use this information in conjunction with the remaining software changes charts.

A6.7. Remaining Software Problems (Unweighted). The following two charts depict the same data in two slightly different formats. The bar chart (figure A6.10) shows stacked bars of remaining software problems, while the line chart (figure A6.11) shows each severity of remaining software problems over time. MEAT version 3.0 presents these charts in color for easy reading.

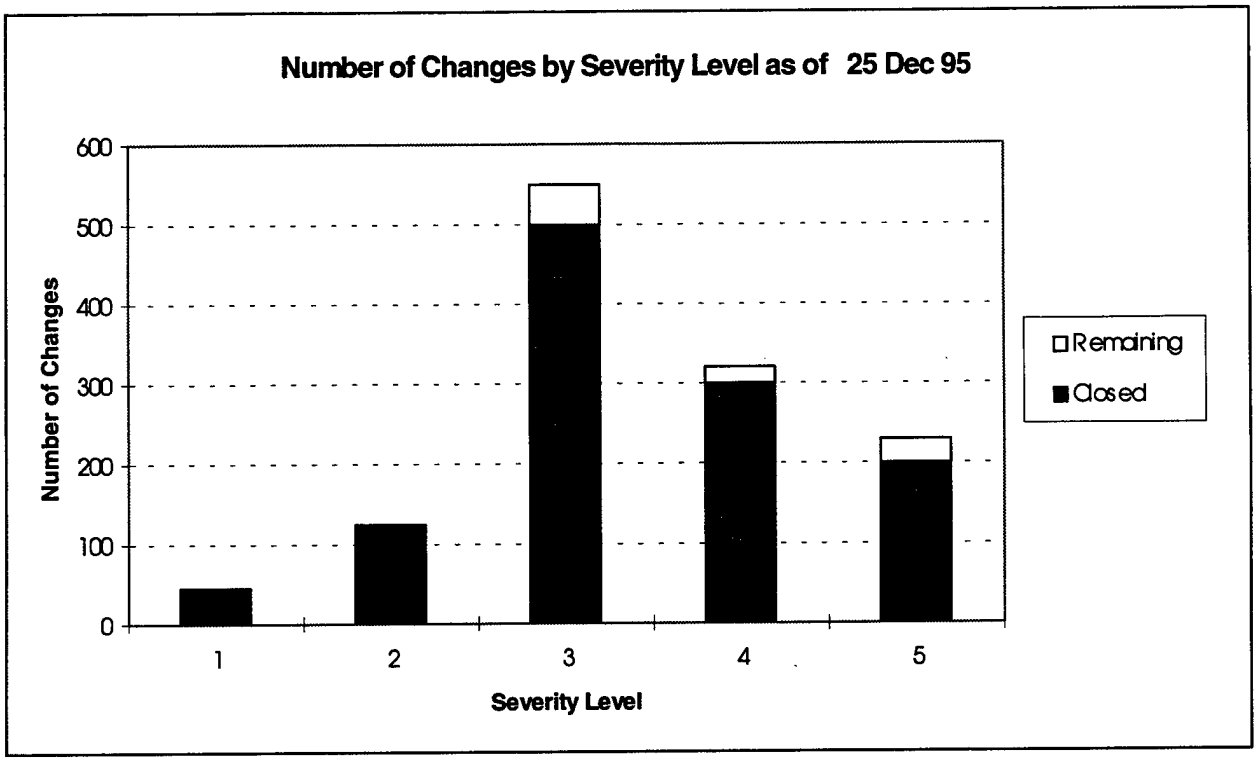


Figure A6.9. Ideal Severity Level Distribution.

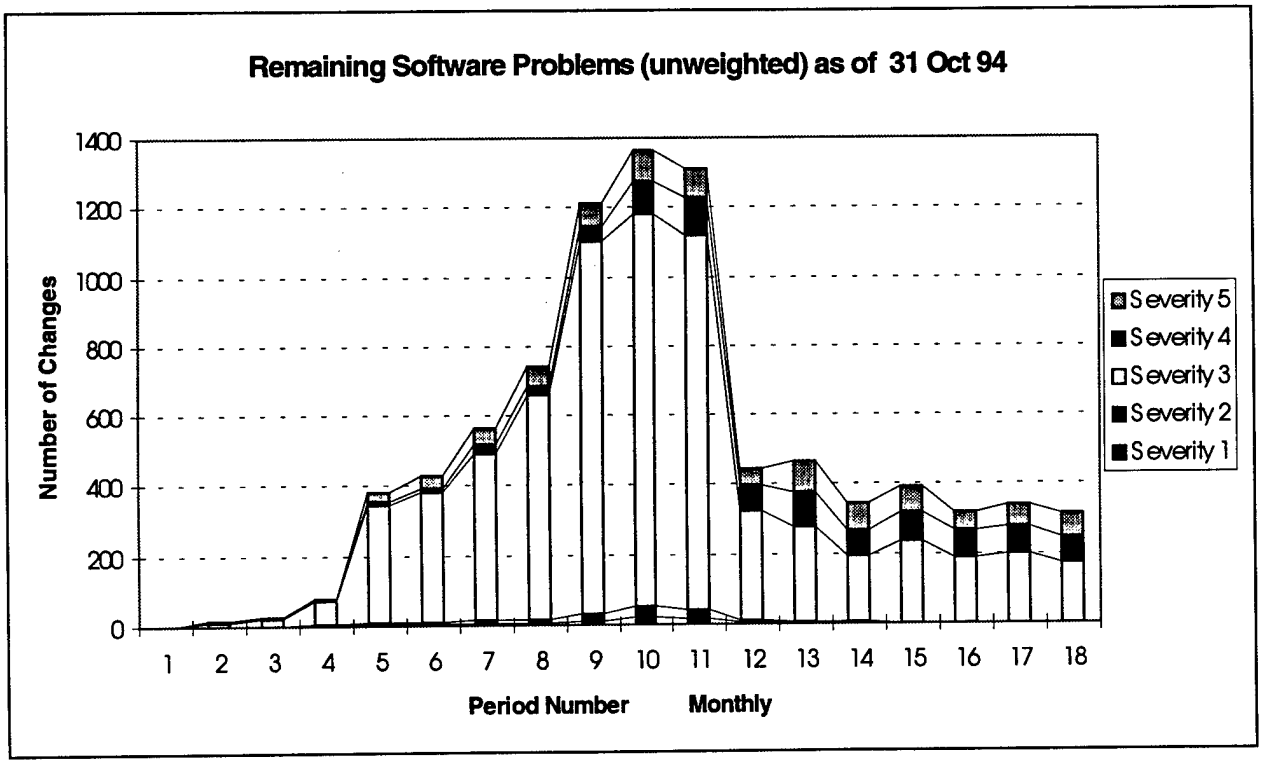


Figure A6.10. Remaining Software Changes (Unweighted) Bar Chart.

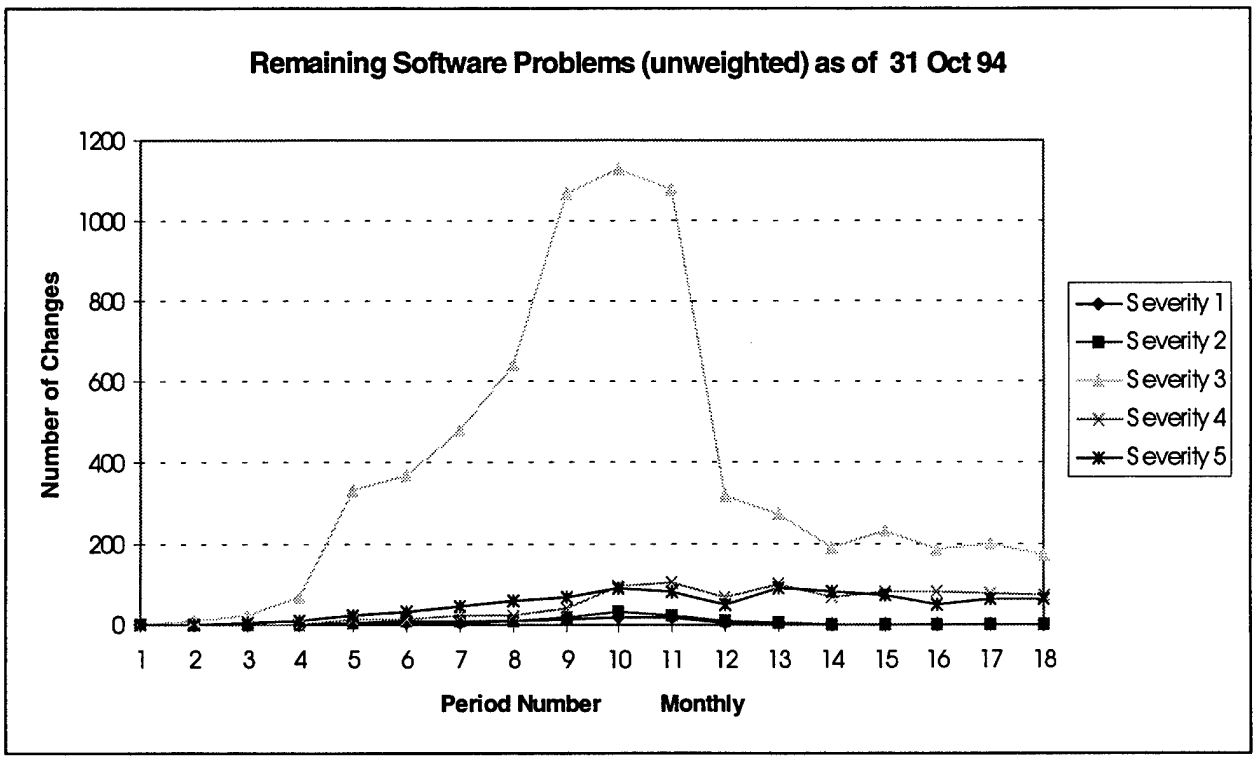


Figure A6.11. Remaining Software Changes (Unweighted) Line Chart.

A6.7.1. **What Are We Looking For?** Just like the remaining changes curves in both *Accumulated Software Changes* charts, we expect the number of remaining software problems to decrease over time. This trend should hold for all severity levels. The sample charts show over 300 remaining software changes in month 18. Nearly 200 of these remaining changes are severity 3. The remaining open changes are severity 4 and 5. The sample also shows no significant downward trend for the last 5 months. This indicates a standing backlog of software changes.

A6.7.2. The ideal remaining software problems bar chart (figure A6.12) shows a shrinking backlog for the entire program. It also shows the distribution across severity levels at each point in time. To easily determine the total number of problems for each severity level, consult the line chart.

A6.7.3. The ideal remaining software problems line chart (figure A6.13) shows a good distribution of problems across severity levels and a shrinking backlog.

A6.8. Total Changes and Change Density by CSCI. Figure A6.14 shows two important pieces of information. First, the bars indicate the total number of changes for each CSCI. Second, the line represents the change density (total number of changes normalized by thousands of lines of code) for each CSCI.

A6.8.1. **What Are We Looking For?** The *Total Changes* bars on the chart show the evaluator which CSCIs generate the bulk of the software change requests for the system. These volatile CSCIs will often be well known to personnel working with the system, but this information is not a complete picture of the problem areas.

A6.8.2. The *Changes/KLOC* (change density) line gives the evaluator the same information normalized by the size of the CSCI. We assume that larger CSCIs will generate more changes, so this metric factors out size. Using these two metrics, the evaluator can begin to identify software maturity drivers from the CSCIs. Given this information, the STM and DSE can focus their efforts on the portions of the program most likely to require future changes.

A6.8.3. Using the bars in figure A6.14, we see that four CSCIs (#8, #17, #7, and #13) represent the bulk of the generated change requests. When we consider total change density (the lines on figure A6.14), we add CSCIs #12, #10, #2, and #4 to the list of CSCIs that drive the maturity of the system.

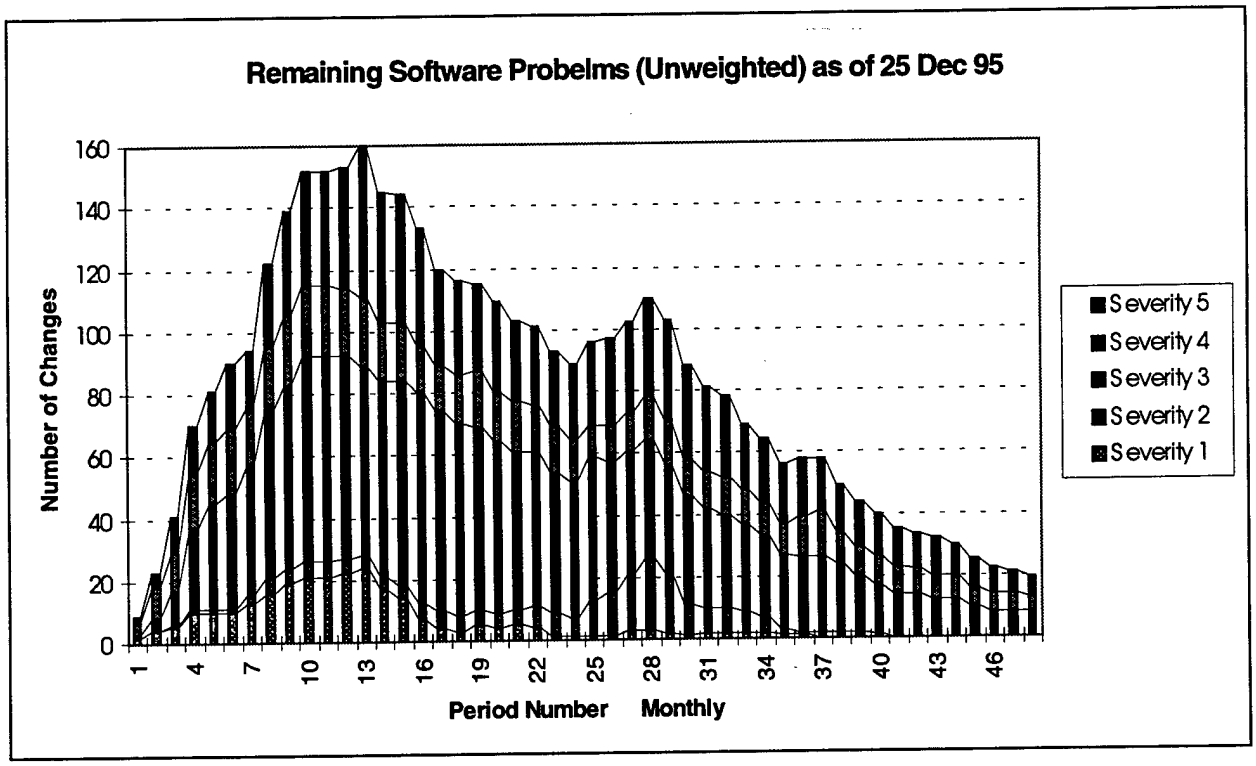


Figure A6.12. Ideal Remaining Software Changes Bar Chart.

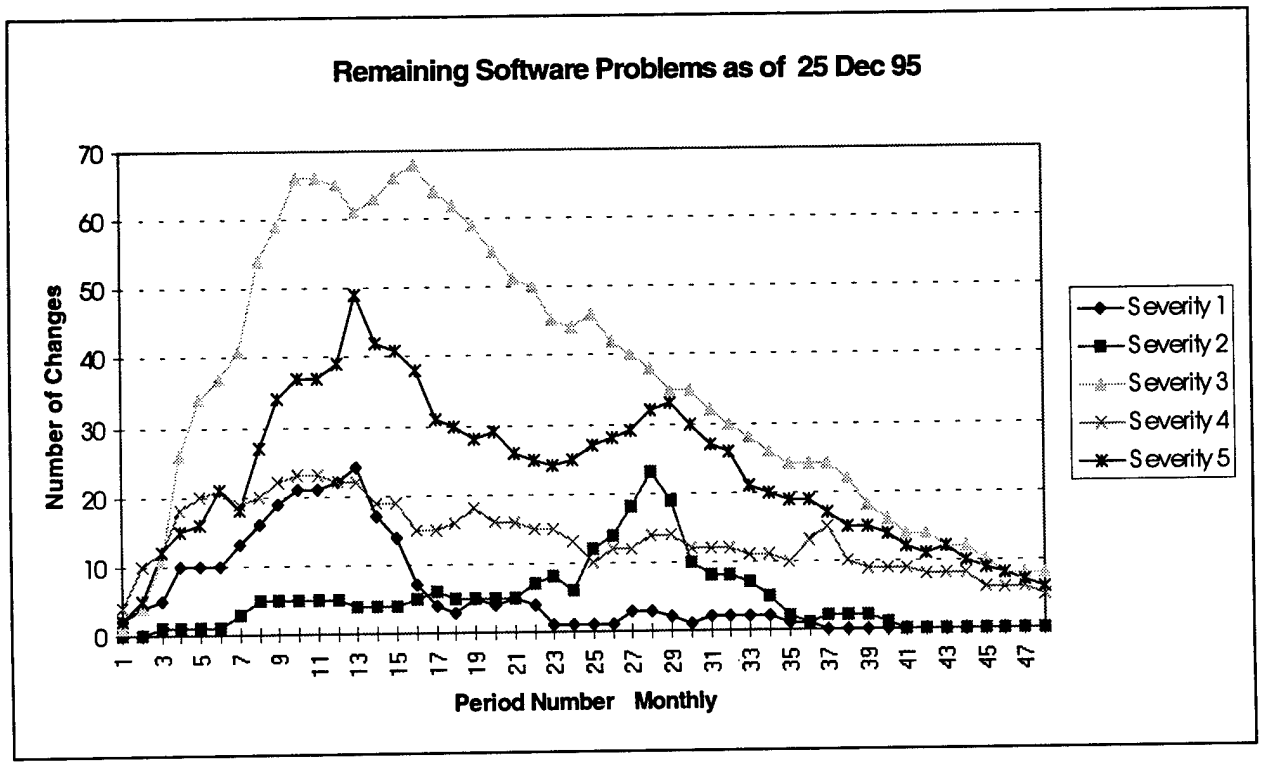


Figure A6.13. Ideal Remaining Software Changes Line Chart.

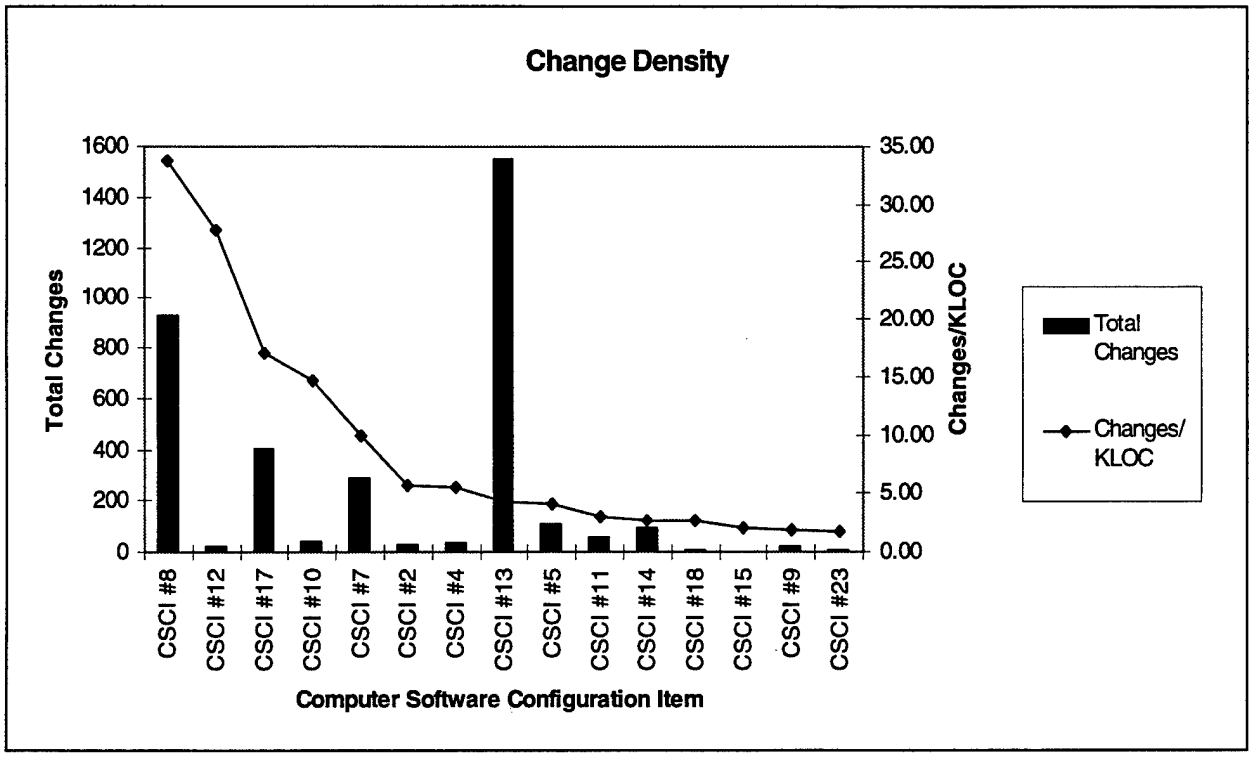


Figure A6.14. Total Changes and Change Density.

A6.9. Remaining Changes and Remaining Change Density by CSCI. Figure A6.15 is similar to the previous chart, but based on remaining change requests. First, The bars indicate the total number of remaining change requests for each CSCI. Second, the line represents the remaining change density (remaining number of changes divided by thousands of new or modified lines of code) for each CSCI.

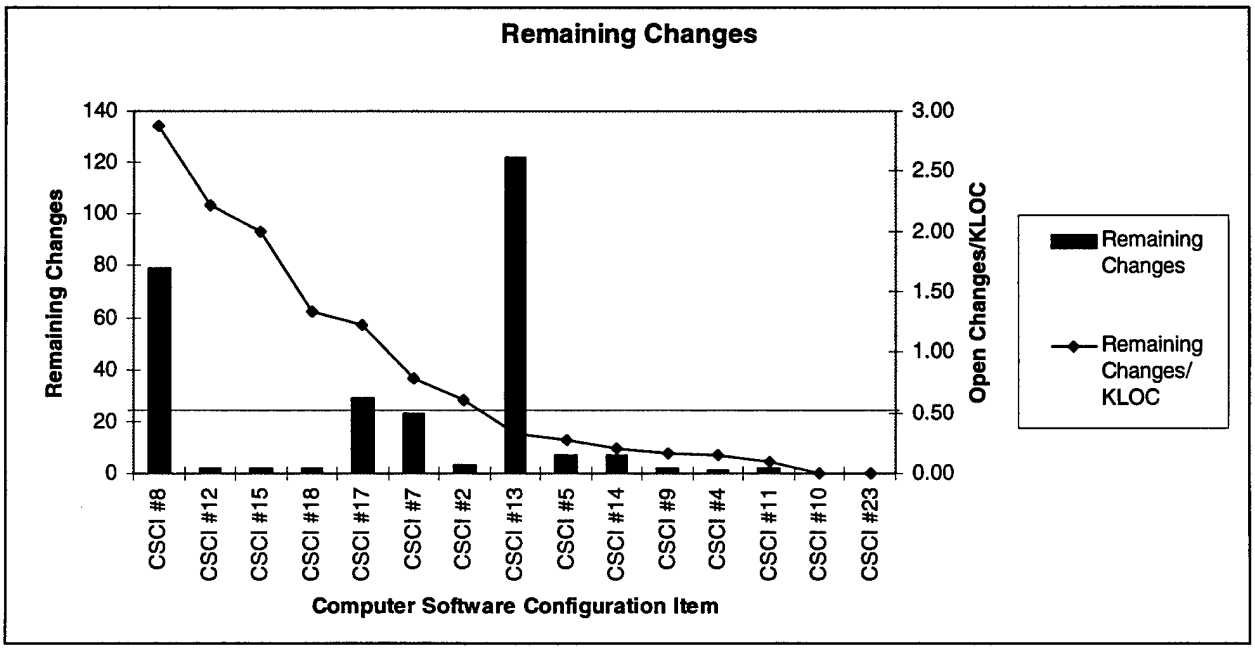


Figure A6.15. Remaining Changes and Remaining Change Density.

A6.9.1. **What Are We Looking For?** The *Remaining Changes* bars on the chart show the evaluator which CSCIs have the most remaining change requests at the current time. As in the previous chart, these problem CSCIs will often be well known to personnel working with the system, but this information is still not a complete picture of the problem areas.

A6.9.2. The *Remaining Changes/KLOC* (defect density) line gives the evaluator the same information divided by the CSCI size (in thousands of lines of new or modified code). We assume that larger CSCIs will generate more changes, so this metric factors out size. AFOTEC has adopted the standard that software is not ready for release until the defect density is below 0.5 (Michael A. Foody, "When is Software Ready For Release?" *UNIX Review*, March 1995). Using these two metrics, the evaluator can begin to identify software maturity drivers from the CSCIs.

A6.9.3. The bars on the sample chart show four problem areas (CSCIs #8, #17, #7, and #13). If you recall, these same CSCIs were identified using the total number of problems chart. The line portion of the chart (remaining changes/KLOC) shows a more distressing problem. Seven of the 15 CSCIs are above the 0.5 changes/KLOC threshold. This indicates these seven CSCIs (CSCIs #8, #12, #15, #18, #17, #7, and #2) are not ready for release (i.e., not yet ready for OT&E).